LA-UR-03-1031

Approved for public release;
distribution is unlimited.

Title:

Further Improvements in ASCI Q Performance and Variability:
an application of the PAL methodology for identifying and eliminating performance variability

Author(s):

Fabrizio Petrini, CCS-3
Darren J. Kerbyson, CCS-3
Adolfy Hoisie, CCS-3

Submitted to:

Los Alamos
NATIONAL LABORATORY

Further Improvements in ASCI Q Performance and Variability:
an application of the PAL methodology for identifying and eliminating performance variability

Fabrizio Petrini, Darren J. Kerbyson, Adolfy Hoisie
Performance and Architecture Lab (PAL)
CCS-3
Los Alamos National Laboratory
February 6th 2003

Executive Summary

This report extends the analysis of the previous CCS-3 report (LAUR-03-0138) on "Identifying and Eliminating the performance variability of the ASCI Q machine" Specifically we have now demonstrated that the performance can be significantly improved using the process we outlined in this previous report

On January 25th and January 27th we carried out a number of optimizations to reduce the noise on QB. In this process we:

- turned off a few daemons that do not perform essential activities,
- correctly configured each cluster domain to confine the noise on the cluster manager (the first node in each cluster)
- selectively configured out nodes that were either slow due to temporary problems (e.g. overheating) or were generating noise that we were not able to eliminate during the tests.

The combined effect of these optimizations is a substantial performance improvement on Sage. In the largest configuration tested (3716 processors), the performance of Sage improved by 55% when compared to previous runs. In addition we have also noted that further improvements should be possible (resulting in an overall improvement of 80% from the current situation representing almost a factor of 2 in speed).

With further optimization we expect to obtain the same results by removing a smaller subset of processors (possibly less than 1%) to achieve near optimum performance.

The net effect of this work will result in a machine in which 99% of the processors can be used without any impact on performance due to noise. Compare this with the current situation in which the best performance is achieved by using 3 processors out of 4 per node (i.e. only 75% of the available processors). Thus the available usable computing power will have been increased by approximately 25% for all applications.

This report outlines the noise reduction process done on January 25th (Section 1), and on January 27th (Section 2). The performance improvement of SAGE is detailed in Section 3, and the generality of the process is detailed in Section 4.

This work would not have been possible without the excellent support given by Ron Green and many others from HP/LANL.

1. Noise Reduction: Part 1 - January 25th

On the January 25th experiment we undertook the following optimizations:

(1) removed about 10 daemons from all nodes (including: envmod, insightd, snmpd, lpd, niff)
(2) decreased the frequency of RMS monitoring by a factor of 2 on each node (from an interval of 30s to 60s)
(3) moved several daemons from nodes 1 and 2 to node 0 on each cluster.

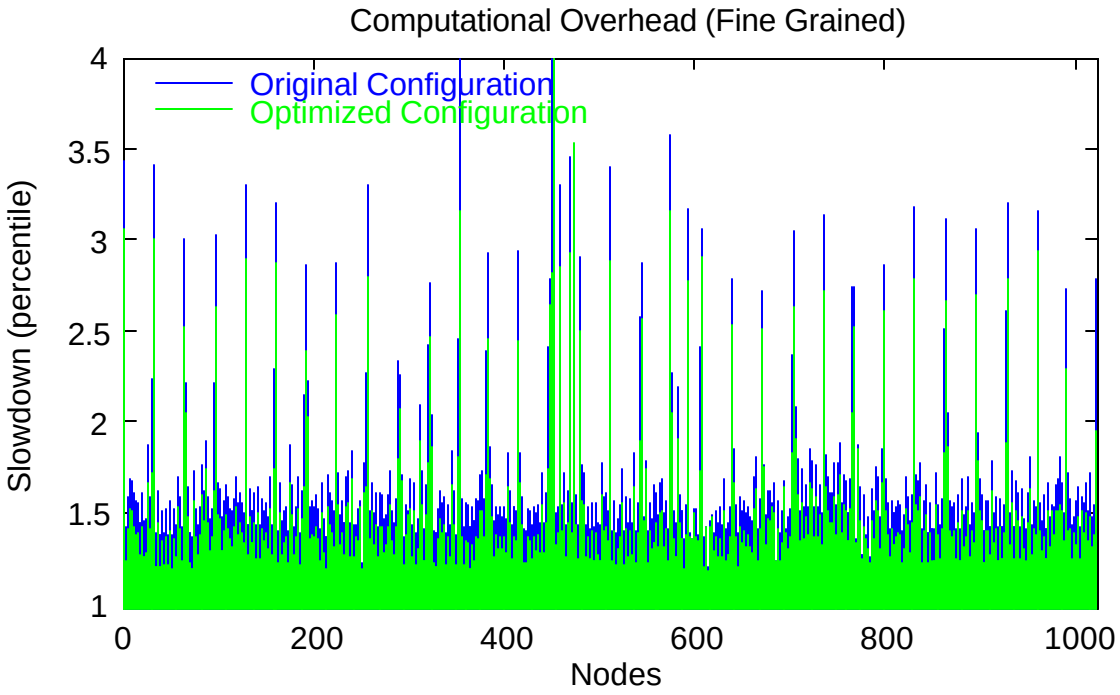

Figure1.1 – Coarse grained noise per node

Figure 1.1 shows the results of the coarse grained measurement before and after the optimizations. In this experiment we simply executed a purely computational benchmark on each processor. Overall, the noise is substantially reduced on all nodes. In particular:

(1) the noise on the compute nodes is reduced by 25% (average),
(2) the noise on the cluster managers (the node 0s in each 32-node cluster) is reduced by 33% (average)
(3) the extra noise on the second node of each cluster (node 1) has been completely eliminated, and now these nodes show the same level of noise of the standard compute nodes.

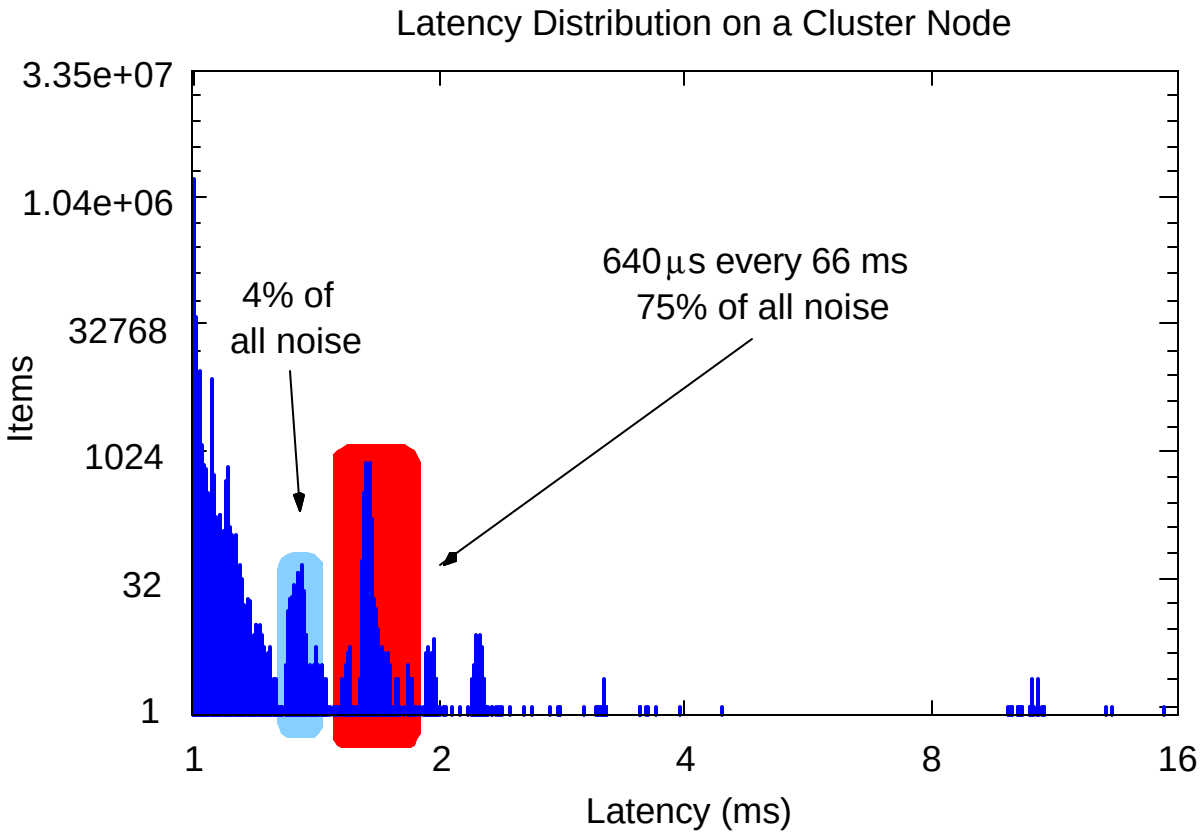

Figure 1.2 Fine grained noise on each node

Figure 1.2 analyzes the noise on a cluster node (nodes 1-30). The graph shows the distribution, after optimizations, of the latency (by latency here we mean duration of each iteration) for a fine-grained benchmark that computes in chunks of 1ms. In an ideal noiseless machine, the graph would have a single bar positioned at 1ms. All the points to the right identify some type of noise or performance degradation. The graph shows that there are still many regular activities (about 10) in each node that interfere with the execution of user applications. Of these activities, one of them takes the lion's share, generating 75% of all noise (indicated in red). Every 66 ms a process in each compute node is interrupted for 640 µs. The regularity of this event (it happens on all nodes with the same frequency and duration) suggests that this is most probably a kernel thread or high priority daemon.

The elimination of this activity is essential to further improve the performance of the Q machine as a whole.

1.2. Barrier Synchronization

A good indicator of the overall level of noise, and the potential slowdown that it can cause on bulk-synchronous applications, is the execution of a simple benchmark where all nodes compute

for a fixed amount of time (that time is the computational granularity), and then synchronize with a global barrier.

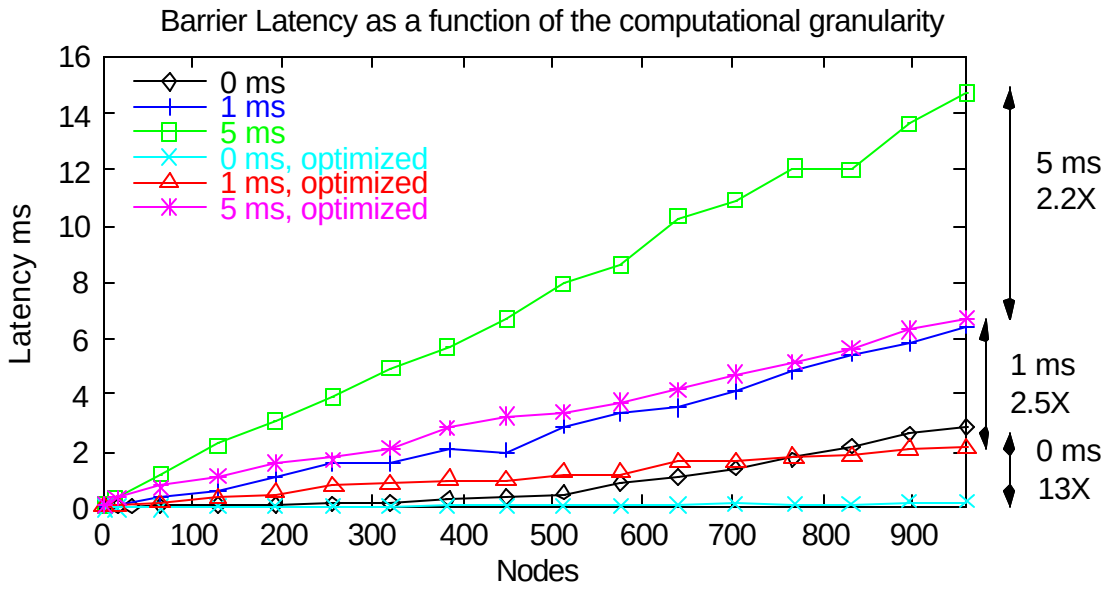

Figure 1.3 – Performance improvement of the barrier synchronization benchmark.

Figure 1.3 shows the results for three types of computational granularity: 0 ms (a simple sequence of barriers without any intervening computation), 1 ms and 5 ms, before and after the optimizations that removed the noise in the machine, as outlined in the previous section.

We can see that with fine granularity (0 ms) the barrier is 13 times faster. The more realistic tests with 1 and 5 ms, which are closer to the actual granularity of actual ASCI codes, show that the performance is more than doubled. This confirms the conjecture that performance variability is closely related to the noise in the nodes.

2. Noise reduction: Part 2 - January 27th

On January 27th we tested more optimizations. The emphasis was on the elimination of nodes that were generating noise, either because of some temporary glitches or because we were not able to completely eliminate the noise inside these nodes.

The noise on node 0 is very high, and it is very likely that it will not be possible to completely eliminate it. A possible solution to this problem is to configure out all the cluster managers of each domain, which amounts to 32 * 4 = 128 processors on each 1024-node segment of the Q machine. A better solution is to dedicate a single processor (or, two, if one is not enough) to the system activities in each cluster manager. We tried to remove a processor per cluster node from the RMS scheduling pool, but we were unable to get this to work in the limited time allotted. So, instead we just configured out node 0 in all clusters.

Quadrics is currently working on the tree-based monitoring data collection that generates noise on node 31 of each cluster, by routing this information directly to the management node. This should result in eliminating the problem with node 31 in the next software release. However, in these experiments we also configured out node 31 in all clusters.

Many nodes in a single cluster were particularly noisy. We were not able to exactly pinpoint the problems, though almost all these nodes were physically located on a single rack that was overheating.

Finally, a specific node was consistently slow, even after several reboots, so we had to configure it out.

In total we had to configure out 95 nodes/380 processors (32 cluster managers + 32 RMS managers + 30 cluster nodes in a single cluster + 1 more node), which amounts to 9.2% of the available computing resources. In the future, we hope to get the same type of optimization by simply configuring out 32 (or 64, if needed) processors, wasting only 1% of the resources. We plan to validate our hypothesis as soon as possible.

3. Sage Scaling: January 27th

The performance of SAGE after removing the noise as identified and discussed in the previous section is shown in Figure 3.1 below. The performance is shown using the cycle-time metric. The performance is also shown for the measurements taken on QA on September 21st, on QB on November 25th, and the new measurements taken on QB on January 27th . Also shown is the expected performance as given by the PAL (CCS-3) performance model.

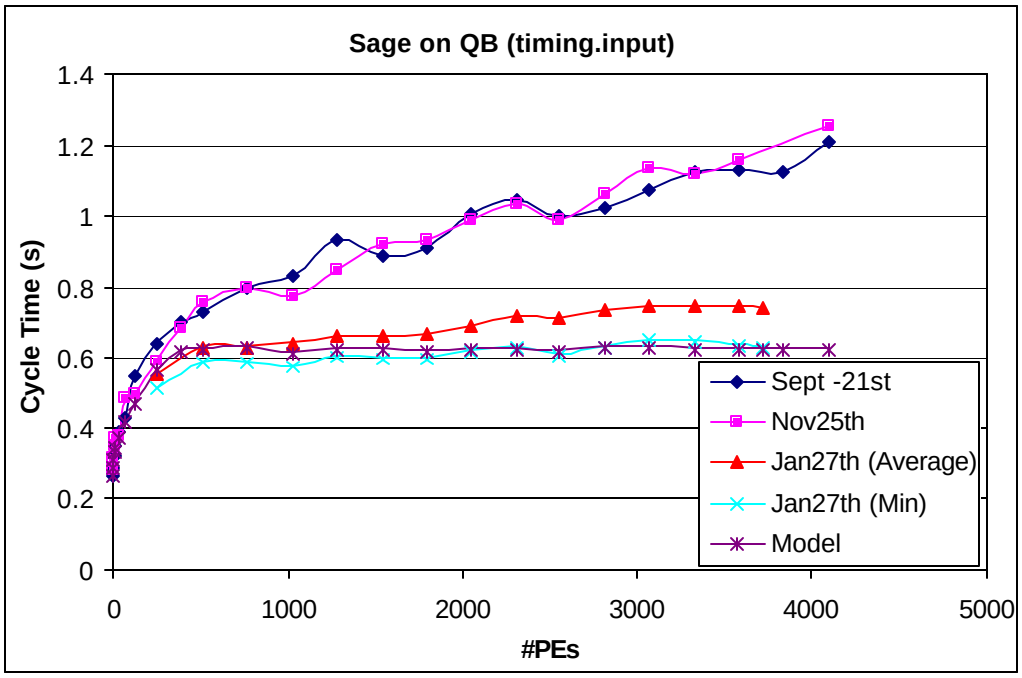

Figure 3.1 – Performance Comparison of SAGE: before and after noise removal.

It can be seen that there is a significant improvement in the application performance as a result of the noise removal. The improvement in the average cycle times is listed in Table 3.1 below for a number of processor counts.

Number Processors	% Improvement after noise removal	% Potential Total Improvement
512	20%	30%
1024	21%	38%
2048	43%	60%
3072	53%	72%
3716	55%	82%

Table 3.1 – Performance improvements on SAGE

Thus the performance of SAGE is now over 55% better on a full 1024 Q segment. Note that the actual performance improvement of SAGE depends on the calculation / problem size being processed.

It is also interesting to note that the minimum cycle time observed after the noise removal is just below our model expectations. This is a clear indication that through the noise removal process we are getting very close to the best performance achievable on QB.

4. Generality of the Results. Conclusions from this Stage of the Work.

In the previous section we have seen how the elimination of a few system activities benefited Sage with a specific input deck. We now try to provide some guidelines to generalize our analysis.

In order to extrapolate the potential gains to other applications, we provide some insight on how the computational granularity of a balanced bulk-synchronous application correlates to the type of noise. The intuition behind this discussion is the following: **while any source of noise has a negative impact on the overall behavior of the machine, a few sources of noises tend to have a major impact on a given application. As a rule of thumb, the computational granularity of the application "enters in resonance" with the noise that has the same magnitude**.

In order to explain this correlation, consider the barrier benchmark of Section 2 for the three optimized configurations with 0, 1 and 5 ms of computational granularity. For each of these cases we analyze the barrier synchronization latency in the largest node count. For example, in such a configuration the barrier takes .19 ms, 2 ms and 7 ms respectively.

For each case we consider the cumulative latency distribution, shown in Figures 4.1, 4.2 and 4.3. Each graph describes how different sources of noise affect the barrier synchronization latency. In Figure 4.1 shows the results for a sequence of barriers without any computation (which represents an extreme case of fine grained computation). We can see that 66% of the delay is caused by the fine grained noise, the one that generates computational holes of less than 4 ms. The heavyweight, but less frequent, noise generated by node 0 in each cluster impacts the barrier latency of only 17%.

With 1ms of computational granularity, the impact of the fine grained noise is only 33% while the relative effect of the heavyweight noise grows to 27%, as shown in Figure 4.2. The primary source of degradation is the medium-grained noise generated by RMS on node 31 and on each cluster node.

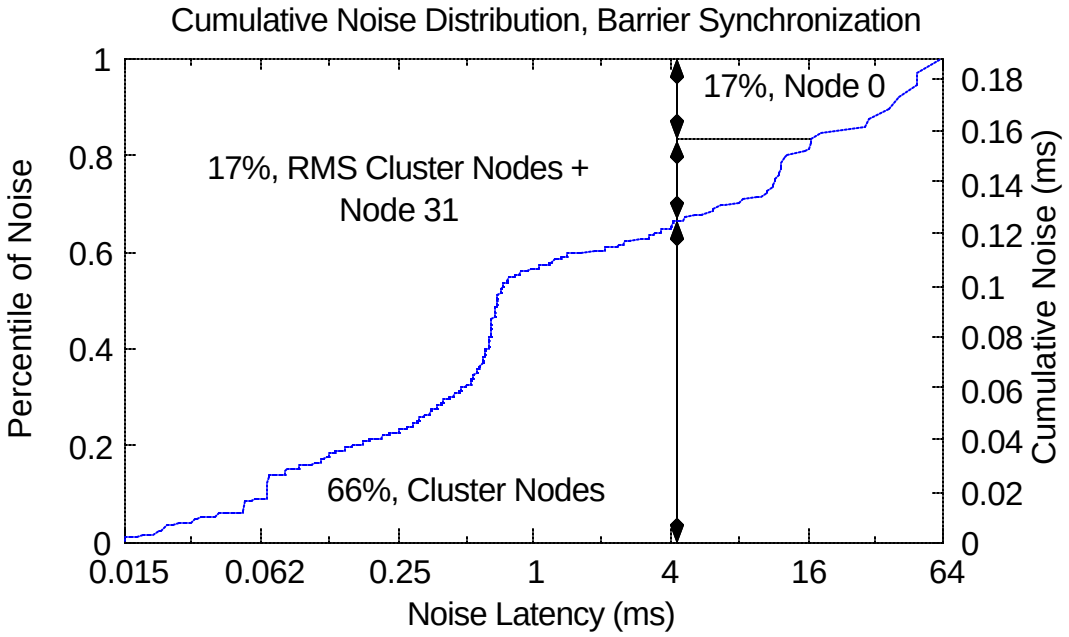

Figure 4.1 – Cumulative noise distribution for a sequence of barrier synchronizations with no intervening computation.

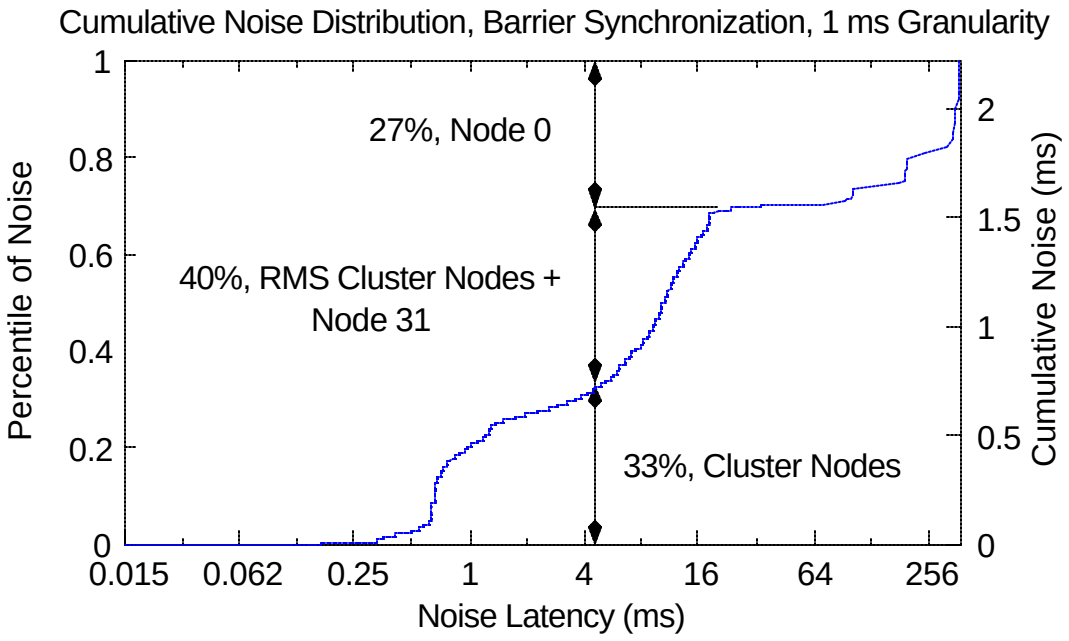

Figure 4.2 - Cumulative noise distribution for a computational granularity of 1 ms.

Finally, we can see in Figure 4.3 that with 5ms more than half of the barrier latency is caused by node 0, while RMS on node 31 plus the cluster nodes takes 33%.

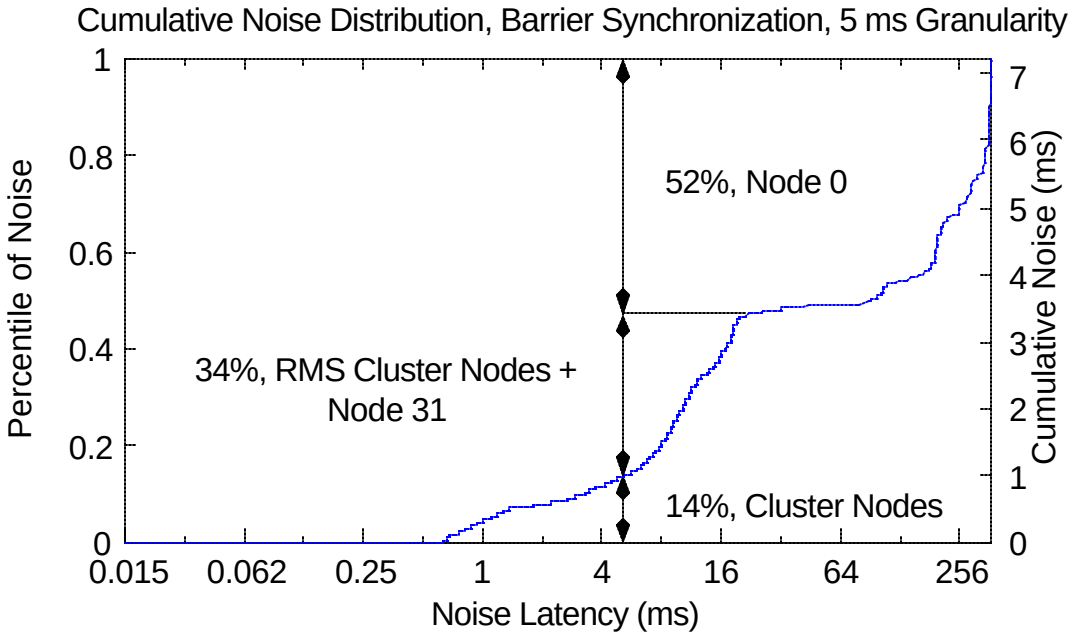

Figure 4.3 - Cumulative noise distribution for a computational granularity of 5 ms.

We can outline the following points, based on the consideration that there is a strong correlation between the computational granularity of an application and the granularity of the noise.

- Load balanced, coarse-grained applications that do not communicate often (as Linpack) will see a performance improvement of a few percent (around 5%) from the elimination of the noise generated by node 0. They are only marginally affected by the other sources of noise.

- We have already shown that fine grained applications, like Sage, can get a substantial performance boost when executed on a large configuration. Sage benefits from the elimination of the medium-weight (RMS on node 31 plus the cluster nodes) and heavyweight noise (node 0).

- Finer grained applications, such as Sn transport codes (Sweep3d) that communicate very frequently with small messages are very sensitive to the fine grained noise that is still present in the machine, as shown in Figure 1.2.